N82-17075

D32

NASA/ASEE SUMMER FACULTY RESEARCH FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

MATHEMATICAL PROGRAMMING TECHNIQUES FOR SCHEDULING SPACELAB CREW ACTIVITIES AND EXPERIMENT OPERATIONS

Prepared By:	Frank H. Mathis
Academic Rank:	Assistant Professor
University and Department:	Baylor University Department of Mathematics
NASA/MSFC:	
Division:	Mission Analysis
Branch	Mission Integration
MSFC Counterpart:	William C. Askew
Date:	August 14, 1981
Contract No.:	NGT 01-006-021 The University of Alabama in Huntsville

MATHEMATICAL PROGRAMMING TECHNIQUES FOR SCHEDULING SPACELAB CREW ACTIVITIES AND EXPERIMENT OPERATIONS

BY

Frank H. Mathis
Assistant Professor of Mathematics
Baylor University

ABSTRACT

The objective of this report is to investigate several mathematical programming techniques which may be applied to the scheduling of experiments and crew activities for Spacelab missions. We will discuss the use of currently known methods, in particular zero-one programming and heuristic dispatching. In addition we propose a new type of scheduling algorithm and present examples to illustrate and test its use.

ACKNOWLEDGEMENT

The author wishes to thank the members of the Mission Integration Branch and, in particular, Bill Askew and Jerry Weiler for their generous support and assistance.

INTRODUCTION

In order to obtain a schedule of activities for a Spacelab mission , a number of experiments must be assigned to specific times in the mission so that a variety of constraints are satisfied. Since this problem may be formulated as a limited resource - project scheduling task, several well-tested algorithms exist for obtaining a good solution.

The use of zero-one programming as developed by Pritsker, Watters and Wolf [4] and improved by several authors (see, for example, [5]) has proved effective on small scheduling problems. However, run time and computer storage requirements increase rapidly with the number of variables present and may prove prohibitive for use on a problem the size of a Spacelab mission.

On the otherhand, dispatching techniques, which take the experiments to be scheduled in a specific order and assign each to the first valid time in the mission, produce results with relatively fast run-time and small storage requirements. However, these methods may fail to yield a good schedule. The result is that a great deal of time must be spent investigating and editing the timeline obtained from a dispatching scheduler to obtain a better schedule.

Clearly there is a need for algorithms which can obtain a good quality timeline and not exceed unreasonable time and storage requirements.

OBJECTIVES

The objective of this report is to investigate current techniques for scheduling with resource-constraints. In particular, we discuss the use of zero-one programming and dispatching methods. We will then introduce a new algorithm which combines characteristics of both existing methods to generate a valid schedule of experiments. Examples and test results are presented to illustrate the feasibility of the use of this algorithm on data typical to Spacelab missions.

DEFINITION OF PROBLEM

In order to formulate the problem of scheduling a Spacelab mission, we assume that each experiment has been divided into a number of individual tasks which we denote as models. A model in turn is divided into separate steps each of which demand a specific set of requirements. In addition, each model may have to be scheduled several times. We denote this by the number of performances of a model.

The constraints imposed on the schedule are of the following types. A step may have several resource requirements. These include the use of a crewman or a certain piece of equipment as well as electrical power, computer memory, data transmission and so forth. Two steps may have a sequencing constraint. That is, step A must be scheduled before step B with specified minimum and maximum delays between the two steps. Each model will have an early and a late start time between which the first step of each performance must be scheduled. Finally, many steps must be scheduled during one of several specific time intervals of the mission. For example, a step may require the ability to view a certain star. Thus that step can only be scheduled during those times in the Spacelab orbit that the star is visible. We refer to this type of requirement as a target. We assume that we know the times at which any target is available during the mission.

Thus the problem is to schedule as many as possible of the specified number of performances of each model such that all constraints are satisfied. We refer to the resulting schedule as a timeline.

As an example of the type problem encountered in a Spacelab mission we will use a data file which was constructed in June 1980 for Spacelab I. We point out that this file is not currently in use; however, we feel it can serve to typify the size and structure of the problem. Table I summerizes this data.

TABLE I

DATA for SPACELAB I (June 1980 version)

Length of timeline	165.61	Hours
Number of models	218	
Number of performances	935	
Total number of steps	3497	
Number of crew available	6	
Types of equipment	38	
Other types of resource constraints	4	
Number to targets	72	

TYPES OF SCHEDULERS

I. Zero-one Programming.

The use of zero-one programming was first applied to scheduling problems in a series of papers by Pritsker, Watters and Wolf [4]. In this approach the timeline is divided into discrete time intervals. Corresponding to each interval a variable must be defined for every step in the problem. A variable is assigned the value of "one" if the corresponding step starts during that time interval and a value of "zero" otherwise. One advantage of this formulation is that the problem is easily represented by a set of linear equalities and inequalities. We illustrate this below. Refer to Table II for a listing of the required variables.

TABLE II

Variables for zero-one programming

i	-	Subscript referring to a model.
j	-	Subscript referring to a performance.
k	-	Subscript referring to a step.
t	-	Subscript referring to a time interval.
x (ijkt)	-	A zero-one variable assigned the value of 1 if step k of performance j of model i starts in time period t, and 0 otherwise.
d (ik)	-	Duration of the kth step of model i.
min (ik)	-	Minimum delay after step k of model i.
max (ik)	-	Maximum delay after step k of model i.
E (i)	-	Early start time for model i.
L (i)	-	Late start time for model i.
N (i)	-	Number of steps for model i.
P (i)	-	Number of performances for model i.

TABLE II

Variables for zero-one programming (cont.)

m	-	Subscript referring to a resource.
r (ikm)	-	Amount of resource m required by step k of model i.
R (mt)	-	Amount of resource m available at time t.

The various constraints may be defined as follows:

Sequencing examples for steps k and k+1 of model i, performance j:

$$\sum_{t=E(i)}^{L(i)} \{t \cdot X(ijkt)\} + d(ik) + \min(ik) \leq \sum_{t=E(i)}^{L(i)} \{t \cdot X(ijk+1t)\}$$

and

$$\sum_{t=E(i)}^{L(i)} \{t \cdot X(ijk+1t)\} \leq \sum_{t=E(i)}^{L(i)} \{t \cdot X(ijkt)\} + d(ik) + \max(ik).$$

Resource example (including targets) for resource m at time $\hat{t}$:

$$\sum_{i} \sum_{j=1}^{P(i)} \sum_{k=1}^{N(i)} \left\{ r(ikm) \sum_{t=\hat{t}-d(ik)}^{\hat{t}} [X(ijkt)] \right\} \leq R(m\hat{t}).$$

In addition we require a step definition constraint for each step. This will assure that the step is assigned to at most one start time.

$$\sum_{t=E(i)}^{L(i)} X(ijkt) \leq 1, \text{ for all } i,j \text{ and } k.$$

Also a performance completion constraint is needed for each performance. This will allow no step in a performance to be scheduled unless the last step is also scheduled.

$$\sum_{t=E(i)}^{L(i)} \sum_{k=1}^{N(i)} X(ijkt) = \sum_{t=E(i)}^{L(i)} \left\{ N(i) \cdot X(ij\ N(i), t) \right\}, \text{for all } i \text{ and } j.$$

Using the above constraints one may program the problem to optimize several objective functions. We list some examples.

To obtain the most steps possible we select the X(ijkt) to maximize.

$$\sum_{i} \sum_{j=1}^{P(i)} \sum_{k=1}^{N(i)} \sum_{t=E(i)}^{L(i)} X(ijkt).$$

Similarly maximizing the following functions produces the most performances, models and experiment time respectively.

$$\sum_{i} \sum_{j=1}^{P(i)} \sum_{t=E(i)}^{L(i)} X(ij\ N(i)\ t),$$

$$\sum_{i} \sum_{t=E(i)}^{L(i)} X(i\ P(i)\ N(i)\ t),$$

$$\sum_{i} \sum_{j=1}^{P(i)} \sum_{k=1}^{N(i)} \sum_{t=E(i)}^{L(i)} \left\{ d(ik) \cdot X(ijkt) \right\}.$$

Since good algorithms exist to handle integer programming with zero-one variables (see, for example, chapter XI of [3]), the above approach appears as an attractive means to solve the scheduling problem. However, if one applies this technique to Spacelab data, obvious problems arise.

First the largest time interval which would reasonably define the timeline is one minute. This means that a formulation of 3497 steps over 165 hours would require well over 3 million variables. Although techniques such as the method of Talbot and Patterson [5] can greatly decrease this storage requirement, the time required to solve the zero-one problem remains substantial. This is because the most efficient zero-one algorithms employ an implicit enumeration technique in which after a feasible solution is obtained, the method back tracks until all possible solutions are directly or indirectly observed. The larger the problem the longer it takes to accomplish the back tracking. In [5] results for test problems with up to 800 zero-one variables and 300 constraints are reported. Using an IBM 370/168 the authors obtain run-times of 6 to 30 seconds CPU. If this were extrapolated to a problem the size of a Spacelab mission, run times of 5 to 30 hours might be expected. These numbers were obtained assuming that the time increases proportionally with size. A more reasonable assumption would dictate that run-time increases with the cube of the size. Indeed this is usually the case for linear programming applications. Extrapolating under this assumption would result in a run-time of over 9000 years!

II. Dispatching Techniques.

Because of the large storage and long run-time requirements of the zero-one formulation, there is a need for alternative methods if one is faced with a problem of any size. The most common approach for scheduling large problems is a dispatching technique. In these methods one places an order on the list of steps to be scheduled, then following this order the steps are assigned to the first time at which all constraints are satisfied. This allows for a schedule to be obtained very rapidly with minimal storage requirements. Clearly, the order on the steps is crucial to the structure of the resulting timeline. In some cases, one may obtain a good timeline by choosing the order based on a heuristic rule or "rule of thumb". For example, one may wish to schedule those steps which require the most resource first. Davis and Patterson [1] have tested several heuristic dispatching rules on various scheduling problems. Also see Grone and Mathis [2] for a discussion of dispatching methods applied to Spacelab missions.

We note that the current scheduler in use at MSFC is a dispatching type program which may use either a random ordering or an order fixed by the operator. We will refer to this program as TLP for Timeline Program. In a test envolving 30 runs of TLP using the random ordering on our sample data, we found that a valid timeline is obtained in less than 2 minutes of elapsed time.

None of the runs produced a timeline with all of the desired models scheduled; the average run scheduled approximately 80% of the models and 87% of the performances.

There are several problems with a dispatching scheduler. There is no guarantee that the resulting schedule is near optional. Also, heuristics seem to be very problem dependent. That is, a rule which works well on a particular problem may prove disastrous for the next problem. Thus, a dispatching scheduler may generate a valid timeline very quickly, but then a considerable amount of time must be spent editing the schedule in order to improve its quality. This is indeed the case with TLP. The timeline produced by TLP can be converted into one in which 93% of the models and 96% of the performances schedule, but only after much investigation and alteration requiring several man-hours of effort.

A NEW ALGORITHM

We now propose a new algorithm which combines some elements of both zero-one and dispatching techniques. We wish to construct a method which has the flexibility of zero-one programming so that after a step is scheduled we may later move the step to another place in the timeline. However, in order to speed up the algorithm, we will not attempt a full back tracking procedure but rather depend on a heuristic rule to dictate which steps should move.

To define our method, we divide constraints into two groups. A constraint is called step dependent if the amount available to a step at a particular time will change as other steps are scheduled at the same time. Crew usage, equipment and other resources are examples of step dependent constraints. Step independent constraints are those which do not depend on how many steps are running at the same time. Examples are early and late start time, sequencing, and targets.

We say that a step is assigned to a feasible time if no step independent constraint is violated, and the timeline is valid if all constraints are satisfied.

The general idea of the algorithm is to start with a feasible timeline and then proceed from the front to the back of the timeline checking each time for validity. If at any time it is found that a step dependent constraint is violated, a particular step scheduled at that time is selected and moved to a future time in the timeline.

Whenever a step is moved, all other steps in the same model must be checked and possibly moved to assure that maximum and minimum delay requirements are satisfied. We will not got into the details of this process but rather denote it by the phrase "move other steps as necessary". If, during an attempt to move step j to a new time, any step in the same model can not be reassigned to a feasible time, we will say that step j cannot be reassigned.

If at any time, a step dependent constraint is violated but no step can be reassigned, then a step is selected and removed from the timeline. In this case all other steps in the same performance must be removed. Of course, the method of choosing a step to be moved or removed is vital to the quality of the resulting timeline. As with a dispatching method, one may use any of several heuristic rules to accomplish this. In the algorithm to follow, we will move the step which uses the most of the violated resource.

This strategy is used in an attempt to cause as many steps as possible to schedule at the present time and to minimize the number of moves required. We do not claim that this rule is necessarily the best.

We refer the reader to table III for a list of variables used in the algorithm. A formal statement of the algorithm follows:

TABLE III

A Variables List for the Scheduling Algorithm

N	-	Number of steps.
st(j)	-	Starting time of step j.
d(j)	-	Duration of step j.
T	-	Current time under investigation for validity.
T (old)	-	Previous time investigated.
T (new)	-	Time to which a stp may be reassigned.
J (T)	-	Set of indices j such that step j is scheduled during time T. That is, $st(j) \leq T \leq st(j) + d(j)$
L	-	Set of indices corresponding to steps for which time reassignments were not possible.
M	-	Number of step dependent constraints.
R (k,T)	-	Amount of resource k available at time T.
r (jk)	-	Amount of resource k required by step j.
S (k,T)	-	Scaled requirement for resource k at time T; $S(k,T) = \left\{ \sum_{j \in J(T)} r(jk) \right\} / R(k,T).$
ϕ	-	The empty set.

ALGORITHM

1. Assign st (j), $j=1,2,\cdots N$, to its earliest feasible start time. Set $T=-1$.

2. Set T (old) = T ; Set $L = \emptyset$.

3. If $\{j \text{ such that st } (j) > T \text{ (old)}\} = \emptyset$, stop ;

   otherwise set $T = \min_{j} \{st(j) \text{ such that } st(j) > T \text{ (old)}\}$.

4. Find all j in J (T).

5. Find k which corresponds to the maximum S (k,T), $k=1,2,\cdots M$.

6. If S $(k,T) \leq 1$, go to 2 ; otherwise continue.

7. If J (T) - L = $\emptyset$, let j be the first index in L, remove step j from the timeline and go to 4; otherwise continue.

8. Find i in J(T)-L to maximize r(ik).

9. Set $T \text{ (new)} = \min_{j} \{st(j) + d(j) \text{ so that j is in } J(T) \text{ and}$

   $e(jk) \neq 0.\}$.

10. Attempt to reassign st(i) to the first feasible time $>$

    T (new); more other steps as necessary.

11. If st(i) can be reassigned go to 4; otherwise, add

    i to L and go to 7.

We point out that because reassignment of a step may cause other steps to move to a time that has already been checked for validity, a single application of the algorithm may not produce a valid timeline. In practice the algorithm must be repeated until no moves are necessary.

EXAMPLES

In order to illustrate the operation of the algorithm, we present several examples. The first is a test problem used in [4]. Also see chapter XII of [3]. The problem contains 6 models, 8 individual steps and 3 types of resources. The data is summerized in Table IV.

TABLE IV
DATA FOR EXAMPLE 1.

Index	Model	Step	Earliest Start	Duration	Latest Start	Resource Requirements		
						1	2	3
1	1	1	0	4	1	5	3	2
2	1	2	4	3	5	0	1	1
3	2	1	0	3	5	2	0	2
4	3	1	1	3	4	1	1	1
5	3	2	4	2	7	2	2	0
6	4	1	1	2	7	2	0	0
7	5	1	2	5	4	2	1	1
8	6	1	2	1	8	1	3	0
Amount of Resource Available at all times						8	5	4

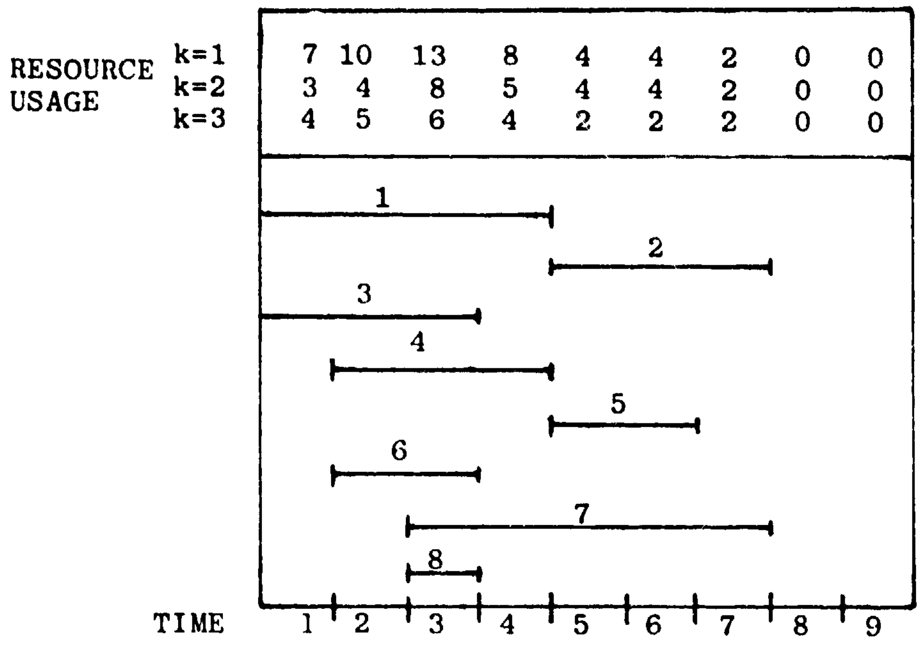
The initial feasible timeline is shown in figure 1. At T=1 the timeline is valid. Moving to T=2 we find resource 1 has the maximum scaled requirement and step 1 is the greatest user. We attempt to reassign step 1 to T(new) = 3; however, this would cause step 2 to be scheduled after its latest start. Hence, step 1 cannot be reassigned. We next attempt to reassign step 3 to T(new). This results in the timeline shown in figure 2.

The timeline is now valid at T=2. We proceed with the algorithm and after six more reassignments we obtain the timeline shown in figure 3. Note that all times are valid.

It is interesting to note that McMillian [3] obtained the schedule shown in figure 4 using zero-one programming. Here the problem is formulated with 33 variables, 37 constraints, and the objective to minimize schedule length. McMillian reports a run-time of 2.3 seconds CPU on an IBM 7044. McMillian also applied a dispatching rule at "minimum slack-time first" to the above example. Slack time is the amount of time between the earliest start time and the latest start time. The result is shown in figure 5. Note that model 3 did not schedule.

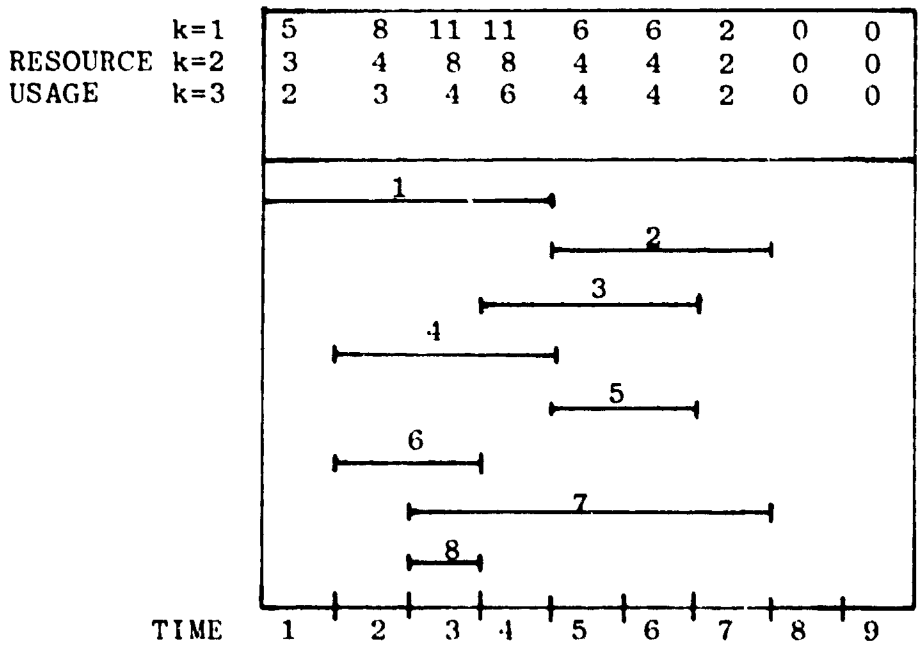
As a second example, a computer program was designed to impliment our algorithm on the DIGITAL VAX-11/780. The first 100 models of the Spacelab data file were selected as an initial test case. This represents a problem with 1911 steps. The algorithm successfully ran in 11 minutes 23 seconds of CPU time. A valid timeline was produced with 1624 steps scheduled. This represents about 85% of the steps requested.

Finally the same program was applied to the full 218 models in the Spacelab data file. Although not all constraints were used - some of the maximum and minimum delays were not checked - the algorithm produced a timeline in 48 minutes of CPU time. Of the 3497 steps requested, 3001 or about 86% were scheduled.



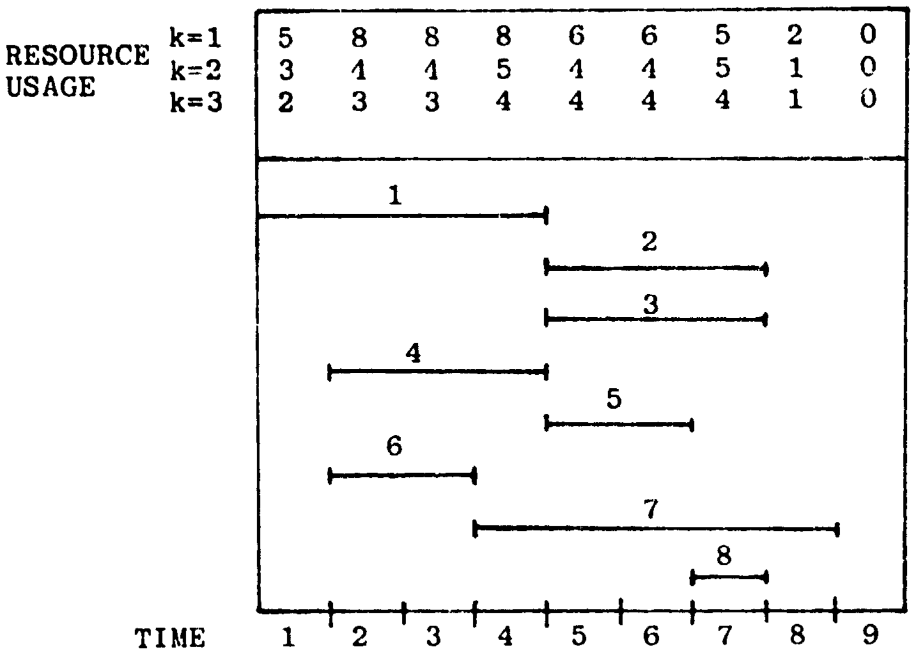
RESOURCE USAGE	1	2	3	4	5	6	7	8	9
k=1	7	10	13	8	4	4	2	0	0
k=2	3	4	8	5	4	4	2	0	0
k=3	4	5	6	4	2	2	2	0	0

Figure 1. Initial Feasible Timeline.



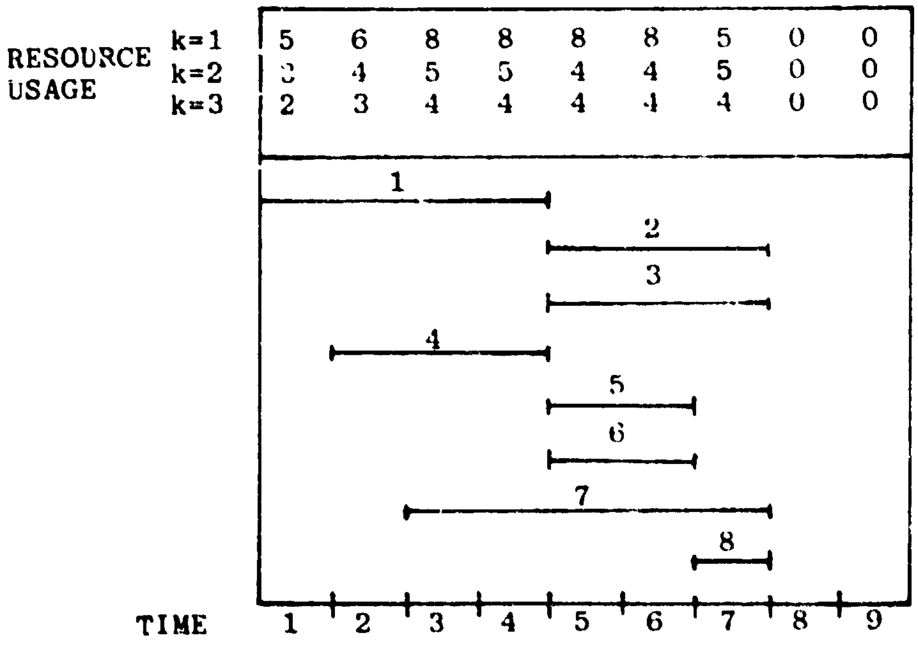
RESOURCE USAGE	1	2	3	4	5	6	7	8	9
k=1	5	8	11	11	6	6	2	0	0
k=2	3	4	8	8	4	4	2	0	0
k=3	2	3	4	6	4	4	2	0	0

Figure 2. Timeline after one move.



RESOURCE USAGE	1	2	3	4	5	6	7	8	9
k=1	5	8	8	8	6	6	5	2	0
k=2	3	4	4	5	4	4	5	1	0
k=3	2	3	3	4	4	4	4	1	0

Figure 3. Final Valid Timeline.



RESOURCE USAGE	1	2	3	4	5	6	7	8	9
k=1	5	6	8	8	8	8	5	0	0
k=2	3	4	5	5	4	4	5	0	0
k=3	2	3	4	4	4	4	4	0	0

Figure 4. Timeline from zero-one method.

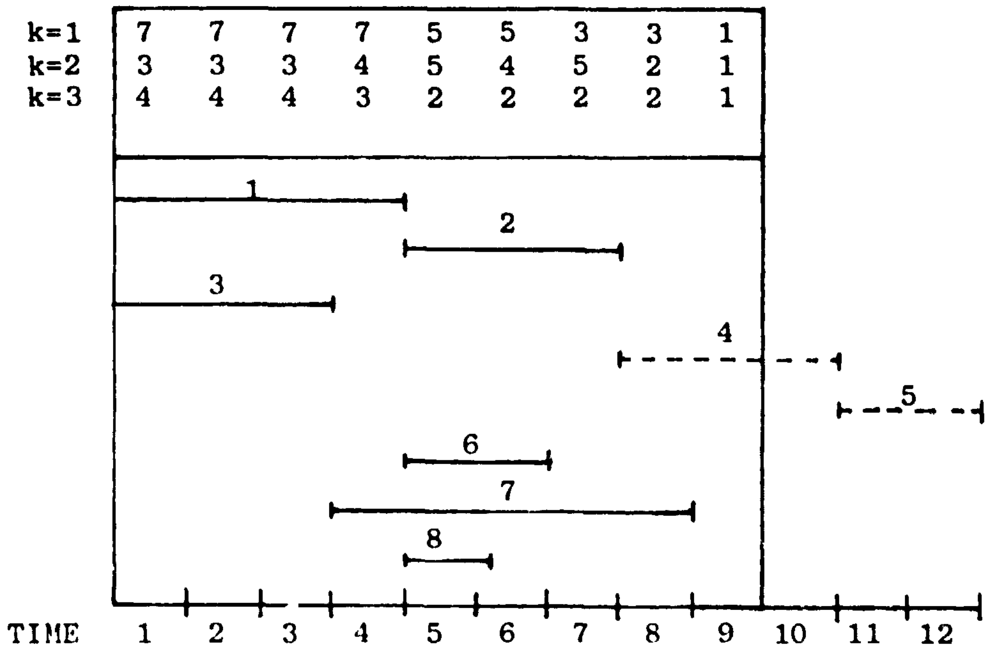


Figure 5. Timeline from dispatching method.

CONCLUSIONS AND RECOMMENDATIONS

Based on our preliminary experimental results, the algorithm presented is feasible for use on Spacelab scheduling problems, although the resulting timeline seems of comparable quality with those obtained by TLP while the expense of run-time is substantially greater than that of TLP. Nevertheless, we feel that the flexibility furnished by the algorithm warrents further investigation at least with respect to two points. First more efficient searching and sorting techniques may be employed in the coding of the algorithm to decrease run-time. More importantly, the use of alternative heuristic rules for the choice of steps to move or remove should be examined. We feel that this investigation could lead to great improvement in the performance of the algorithm as well as the quality of the resulting timeline.

REFERENCES

1. Davis, E. W. and J. H. Patterson, " A Comparison of Heuristic and Optimum Solutions in Resource - Constrained, Project Scheduling", Management Science, Vol. 21 (1975), pp. 944-955.

2. Grone, R. D. and F. H. Mathis, "A Ranking Algorithm for Spacelab Crew and Experiment Scheduling", NASA CR-161511, Marshall Space Flight Center, October 1980.

3. McMillan, C., Mathematical Programming, John Wiley and Sons, Inc., New York, 1970.

4. Pritsker, A., L. Watters and M. Wolf, "Multiproject Scheduling with Limited Resources: A Zero-one Programming Approach", Management Science, Vol. 16 (1969), pp. 93-108.

5. Talbot, F. B. and J. H. Patterson, "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource - Constrained Scheduling Problems", Management Science , Vol. 24 (1978), pp. 1163-1174.